

METHOD AND APPARATUS FOR ENCRYPTING DATA IN A WIRELESS COMMUNICATION SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

- [0001] This application is a continuation of U.S. Patent Application Serial No. 10/434,749 filed May 9, 2003, which is a continuation of U.S. Patent No.10/081,750, filed February 21, 2002, which is a continuation of U.S. Patent No. 6,385,316, issued May 7, 2002, which is a continuation of U.S. Patent No.6,075,859, issued June 13, 2000, all assigned to the assignee hereof and hereby expressly incorporated herein.

BACKGROUND OF THE INVENTION

I. Field of the Invention

- [0002] The present invention relates to communications systems. More particularly, the present invention relates to a novel and improved method for encrypting data for security in wireless communication systems.

II. Description of the Related Art

- [0003] In a wireless communication system, it is desirable for the service provider to be able to verify that a request for service from a remote station is from a valid user. In some current cellular telephone systems, such as those deploying the AMPS analog technology, no provision is made to deter unauthorized access to the system. Consequently, fraud is rampant in these systems. One fraudulent means for obtaining service is known as cloning, in which an unauthorized user intercepts the information necessary to initiate a call. Subsequently, the unauthorized user can program a mobile telephone using the intercepted information and use that telephone to fraudulently receive telephone service.
- [0004] To overcome these and other difficulties, many cellular telephone systems have implemented authentication schemes such as that standardized by the Telecommunications Industry Association (TIA) in EIA/TIA/IS-54-B. One facet of this authentication scheme is encryption of information, transmitted over the air, that is required to receive service. This information is encrypted using the Cellular Message

Encryption Algorithm (CMEA). The CMEA algorithm is disclosed in U.S. Patent No. 5,159,634, entitled "CRYPTOSYSTEM FOR CELLULAR TELEPHONY", incorporated by reference herein.

[0005] Several major weaknesses have been discovered in CMEA which allow encrypted information to be deciphered using current standard computational equipment in a relatively short period of time. These weaknesses will be thoroughly outlined hereinafter followed by a description of the present invention which overcomes these weaknesses. CMEA has been published on the Internet, hence these weaknesses are open for discovery by anyone with an interest in doing so. Thus, a new algorithm for encryption is desirable to replace CMEA to avoid the interception and fraudulent use of authentication information necessary to initiate cellular service.

SUMMARY OF THE INVENTION

[0006] The present invention is a novel and improved method for data encryption. The present invention is referred to herein as Block Encryption Variable Length (BEVL) encoding, which overcomes the identified weaknesses of the CMEA algorithm. The preferred embodiment of the present invention has the following properties:

- Encrypts variable length blocks, preferably at least two bytes in length;
- Self-inverting;
- Uses very little dynamic memory, and only 512 bytes of static tables;
- Efficient to evaluate on 8-bit microprocessors; and
- Uses a 64 bit key, which can be simply modified to use a longer or shorter key.

[0007] The first weakness identified in CMEA is that the CAVE (Cellular Authentication Voice Privacy and Encryption) table used for table lookups is incomplete. It yields only 164 distinct values instead of 256. The existence of a large number of impossible values makes it possible to guess return values of tbox() or key bytes, and verify the guesses. This first weakness is mitigated in the present invention by replacing the CAVE table with two different tables chosen to eliminate the exploitable statistical characteristics of the CAVE table. These tables, called t1box and t2box, are strict permutations of the 256 8-bit integers, where no entry appears at its own index position. In addition, t1box[i] does not equal t2box[i], for all values of i.

These two tables were randomly generated with candidates being discarded which did not meet the above criteria.

[0008] The second weakness of CMEA is the repeated use of the value of a function called `tbox()`, evaluated at zero. The value `tbox(0)` is used twice in the encryption of the first byte. This makes it possible to guess `tbox(0)` and use the guess in determining other information about the ciphering process, notably the result of the first step of CMEA for the last byte, and the arguments of the two values of `tbox()` used in encrypting the second byte. It also makes it possible, through a chosen-plaintext attack, to determine `tbox()` by trying various plaintext values until a recognized pattern appears in the ciphertext. This second weakness is mitigated by changing the self-inverting procedures used in CMEA to a preferred set of procedures providing better mixing. This is done by introducing a second pass using a different table (`t2box`). In this situation there are two values of `tbox()` derived from different tables with equal significance which serve to mask each other.

[0009] A related weakness in CMEA is that information gathered from analyzing texts of different lengths can generally be combined. The use of the second critical `tbox()` entry in BEVL depends on the length of the message and makes combining the analysis of different length texts less feasible.

[0010] A third weakness discovered in CMEA is incomplete mixing of upper buffer entries. The last $n/2$ bytes of the plaintext are encrypted by simply adding one `tbox()` value and then subtracting another value, the intermediate step affecting only the first half of the bytes. The difference between ciphertext and plaintext is the difference between the two values of `tbox()`. BEVL addresses this third weakness by performing five passes over the data instead of three. The mixing, performed by CMEA only in the middle pass, is done in the second and fourth passes which mix data from the end of the buffer back toward the front. The middle pass of CMEA also guarantees alteration of at least some of the bytes to ensure that the third pass does not decrypt. In an improved manner, BEVL achieves this goal in the middle pass by making a key dependent transformation of the buffer in such a way that at most a single byte remains unchanged.

[0011] CMEA's fourth weakness is a lack of encryption of the least significant bit (LSB) of the first byte. The repeated use of `tbox(0)` and the fixed inversion of the LSB in the second step of CMEA results in the LSB of the first byte of ciphertext being

simply the inverse of the LSB of the first byte of plaintext. BEVL avoids this fourth weakness through a key dependent alteration of the buffer during the middle pass which makes the LSB of the first byte unpredictable on buffers of two bytes or more in length.

[0012] A fifth weakness of CMEA is that the effective key size is 60 rather than 64 bits. As such, each key is equivalent to 15 others. BEVL increases the number of table lookups while decreasing the number of arithmetic operations, ensuring that all 64 bits of the key are significant.

[0013] Finally, CMEA's `tbox()` function can be efficiently compromised by a meet-in-the-middle attack. Once four `tbox()` values are derived, the meet-in-the-middle attack can be accomplished with space and time requirements on the order of 2^{30} , independent of the composition of the CAVE table. BEVL addresses this in a number of ways. The construction of the `tbox()` function recovers two unused bits of the key. The repetition of the combination with the least 8 bits of the encryption key at both the beginning and end of `tbox()` means that the minimum computation and space should be increased by eight bits. Since there are two sides of each table, and two different tables, the minimum complexity should be increased by another two bits, leading to a minimum space and time requirement on the order of 2^{42} . Further, the meet-in-the-middle attack on CMEA requires the recovery of at least some of the `tbox()` entries. This is made more difficult using BEVL, which requires simultaneous attacks on two separate sets of `tbox()` values, which tend to disguise each other.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify correspondingly throughout and wherein:

[0015] FIG. 1 is a block diagram illustrating the encryption system of the present invention;

[0016] FIG. 2 is a flow diagram of an exemplary embodiment of the method of encrypting a block of characters in the present invention;

[0017] FIG. 3 is a "C" program implementing the exemplary embodiment of the method of encrypting a block of characters in the present invention;

[0018] FIG. 4 is an exemplary embodiment of t1box; and

[0019] FIG. 5 is an exemplary embodiment of t2box.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0020] The exemplary embodiment of the present invention consists of a first station **1000** encrypting data for wireless transmission to a second receiving station **2000**, as depicted in FIG. 1. First station **1000** can be a remote station transmitting to a second station **2000**, which could be a base station. Alternatively, first station **1000** could be a base station transmitting to a second station **2000**, which could be a remote station. In all likelihood, both remote and base stations will have encryption and decryption means, as well as transmission and reception means, but the simplified system shown in FIG. 1 shows clearly the elements required to enable the present invention. Further, the benefits of this invention are not limited to wireless communications but can be readily applied in any situation where secure data must be transmitted over a medium which is susceptible to interception, as will be well understood by those skilled in the relevant art.

[0021] In FIG. 1, memory **10** containing the necessary data for encryption according to the BEVL algorithm of the present invention is connected with processor **20**. In the exemplary embodiment, processor **20** is a relatively simple 8-bit microprocessor, capable of executing instructions stored in BEVL code **19**. Processor **20** contains an arithmetic logic unit (ALU, not shown) capable of performing simple 8-bit instructions such as bitwise exclusive OR (referred to simply as XOR or denoted \approx hereinafter), integer addition and subtraction, and the like. Processor **20** is also capable of general program flow instructions and the ability to load and store values from a memory, such as memory **10**. Those skilled in the art will recognize that these requirements are quite minimal, making the present invention quite suitable to applications where size and/or cost requirements make simple microprocessors desirable, such as in portable devices. Clearly the present invention can easily be implemented using more powerful microprocessors as well.

[0022] Memory **10** contains tables t1box **12** and t2box **14**, an encryption key **16**, and the code to be executed (BEVL code) **19**. Data to be encrypted is input to processor **20**, which stores that data in memory **10** in a location referred to as data **18**. Although FIG.

1 depicts all these elements in a single memory, it is understood that a plurality of memory devices could be used. In the preferred embodiment, the tables 12 and 14 as well as BEVL code 19 are stored in non-volatile memory such as EEPROM or FLASH memory. These portions of the memory need not be writeable.

[0023] Encryption key 16 can be generated by a number of means that are well known in the art. A simple embodiment may have key 16 in non-volatile memory that is programmed once at the time the station is activated for service. In the exemplary embodiment, key 16 is generated and changed according to the protocol as set forth in the aforementioned EIA/TIA/IS-54-B.

[0024] The data to be encrypted, data 18, is stored in random access memory (RAM). The encryption will be performed "in place", which means the memory locations holding the unencrypted data at the beginning of procedure will also hold the intermediate values as well as the final encrypted data.

[0025] Data 18 is encrypted in processor 20 according to BEVL code 19, utilizing t1box 12, t2box 14, and encryption key 16. A description of the encryption process is detailed hereinafter.

[0026] Encrypted data 18 is delivered by processor 20 to transmitter 30 where it is modulated, amplified and upconverted for transmission on antenna 40.

[0027] Antenna 50 receives the data and passes it to receiver 60 where the data is downconverted, amplified, demodulated, and delivered to processor 70. In the exemplary embodiment, the format for the wireless communication between the two stations depicted in FIG. 1 is described in "Mobile Station-Base Station Compatibility Standard for Dual-Mode Wide Band Spread Spectrum Cellular System", TIA/EIA/IS-95-A. The use of CDMA techniques in a multiple access communication system such as a wireless telephone system is disclosed in U.S. Patent No. 4,901,307, entitled "SPREAD SPECTRUM MULTIPLE ACCESS COMMUNICATION SYSTEM USING SATELLITE OR TERRESTRIAL REPEATERS," assigned to the assignee of the present invention, and incorporated by reference herein. The use of CDMA techniques in a multiple access communication system is further disclosed in U.S. Patent No. 5,103,459, entitled "SYSTEM AND METHOD FOR GENERATING SIGNAL WAVEFORMS IN A CDMA CELLULAR TELEPHONE SYSTEM," also assigned to the assignee of the present invention, and incorporated by reference herein.

[0028] Processor 70, which has the same requirements as processor 20, is coupled to memory 80. Memory 80 is comprised of memories 82, 84, 86, 88, and 89 which are analogous to memories 12, 14, 16, 18, and 19, respectively. Processor 70 stores the encrypted data in data memory 88. Key 86 is determined in like fashion to key 16, described previously. Tables 82 and 84 are identical to tables 12 and 14. Since the data processing in this invention is self-inverting, BEVL code 89, identical to BEVL code 19, is executed in processor 70 in conjunction with t1box 82, t2box 84, and key 86 on encrypted data 88, just as was done in the encryption process of data 18. As before, the data processing is performed “in place”, and the result in data 88 will be the decrypted data. Processor 70 retrieves the decrypted data from memory 80 and delivers it for subsequent use through the data output. In the exemplary embodiment, the resultant data will be used in authentication procedures as disclosed in EIA/TIA/IS-54-B.

[0029] FIG. 2 illustrates a flow chart of the method used by processors 20 and 70 in conjunction with previously described memory elements 10 and 80, respectively. As mentioned previously, the encryption process is self-inverting, meaning the decryption process is the same as the encryption process. Hence, only the encryption process will be described in detail. The decryption process will be obvious by substituting the encrypting blocks of FIG. 1 with the analogous decrypting blocks of FIG. 1 as set forth previously.

[0030] Block 99 marks the beginning of the encryption process. An array of characters named buf[] is used to describe the characters to be encrypted as stored in data memory 18. The variable n denotes the length of the message to be encrypted in terms of number of characters. As stated previously, one of the improvements present in the BEVL process is the five pass encryption that takes place. Each of the five passes has been blocked out in dashed lines and labeled 1 - 5 to make them easy to distinguish. Each pass has notable similarities and differences. Passes 1, 3, and 5 use the table t1box 12 and work from the beginning of the buffer towards the end. Passes 2 and 4 use the table t2box 14 and work from the end of the buffer until the beginning is reached. BEVL's self-inverting property comes from the fact that pass 3 is self-inverting, while pass 1 is the inverse of pass 5 and pass 2 is the inverse of pass 4.

[0031] In the preferred embodiment of the present invention, the passes are made in opposite directions. In alternative embodiments, passes could progress in the same

direction, with alternating passes using the same or different tables (re-using the same table in multiple passes does make the encryption more robust, but not as robust as when different tables are used). Inserting additional passes is another alternative which can be used in combination with either approach. In the situation where passes are made in the same direction, modifications to the first buffer entry are more predictable, with predictability decreasing in modifications further down the buffer. When alternating opposite pass directions are used, the modification to the first byte in the buffer is fairly predictable. However, the modification to that byte in the second pass depends on all the bytes in the buffer, making it much less predictable. In similar fashion, the modification to the last byte in the buffer depends on all the bytes in the buffer during the first pass, while a more predictable change is made in the second. Since the predictability of change is distributed more evenly using passes in opposite directions, doing so is much preferable to using multiple passes in the same direction. Note that pass 3 doesn't really have a direction, since the change made would be identical either way.

[0032] In each pass, a function `tbox()` is used. It is in this function that key **16** is incorporated. The parameters passed to function `tbox()` consist of a 256 byte table which will either be passed `t1box 12` or `t2box 14`, and an index labeled `tv`. In the exemplary embodiment, `tbox()` is defined as:

$$\text{tbox}(B, tv) = B[B[B[B[B[B[B[B[B[tv \approx k0] + k1] \approx k2] + k3] \approx k4] + k5] \approx k6] + k7] \approx k0], \quad (1)$$

where

`k0` through `k7` denote eight 8-bit segments which when concatenated form the 64-bit key **16**;

`B[x]` is the `x`th 8-bit element of an array `B`;

`≈` denotes the bit-wise exclusive OR operation; and

`+` represents modulo 256 addition.

In an alternative embodiment, where a key of a certain length provides encryption that is considered too strong, the key strength can be artificially limited without changing the length of the key by altering the `tbox()` function. For example, a 64 bit key can be

artificially limited to 40 bits by using the 64 bit key in such a manner that it is in an equivalence class of 2^{24} others while still ensuring that any single bit change to the key will produce a different result. The following definition of `tbox()` exhibits the recommended variation to render a 64 bit key effectively a 40 bit key:

$$\begin{aligned} \text{tbox}(B, tv) = & B[B[B[B[B[B[B[B[tv \approx k0] + k1] \approx (k2 \approx k3)] + (k2 \approx k3)] \\ & \approx (k4 \approx k5)] + (k4 \approx k5)] \approx (k6 \approx k7)] + (k6 \approx k7)] \approx k0], \end{aligned} \quad (2)$$

where

$k0$ through $k7$ denote eight 8-bit segments which when concatenated form the 64-bit key **16**;

$B[x]$ is the x th 8-bit element of an array B ;

\approx denotes the bit-wise exclusive OR operation; and

$+$ represents modulo 256 addition.

[0033] The `tbox()` function is designed such that each of the intermediate operations are permutations, meaning each input has a one-to-one mapping to an output. In the exemplary embodiment, the operations used are modulo 256 addition and logical exclusive OR. If the input value passed to `tbox()` is a permutation, and the table lookup is as well, the use of these functions guarantees that the output of `tbox()` will also be a one-to-one function. In other words, the `tbox()` function as a whole is guaranteed to be a permutation if the table passed to it also is. This is not the case for CMEA, where the steps in the `tbox()` function are not one-to-one. Therefore, in CMEA, even if the CAVE table, which is not a permutation, were to be replaced with a table which is a permutation, the output of `tbox()` still would not be a permutation. Conversely for BEVL, any choice of one-to-one functions for combining key material to generate the final permutation would be acceptable. The exemplary embodiment is one such method. Alternative methods can easily be substituted by those skilled in the art which still conform to this permutation principle of the present invention. Intermediate functions which do not preserve the one-to-one nature of the output can alternatively be employed in the BEVL `tbox()` function, but the results would be sub-optimal.

[0034] A further improvement included in the definition of `tbox()` is that some of the key bits are used both at the beginning and at the end. In the exemplary embodiment key byte `k0` is used, but alternative embodiments can employ any of the key bits and accomplish the same improvement. The use of the same value defeats the meet-in-the-middle attack. Failing to reuse at least some of the key information at both the beginning and end allows a straightforward, albeit computationally complex, derivation of the key from a small number of values of the `tbox()` function. With this reuse, tables used in efforts to attack the encryption require much more space and computations required to find a solution are much more extensive.

[0035] The exemplary embodiment of BEVL details the use of the `tbox()` function in conjunction with the two tables `t1box` and `t2box`. The resultant outputs are key-dependent permutations of the possible inputs. However, since the values of the function depend only on the key, not on the data, the function can alternatively be pre-computed for the 256 possible inputs and two possible tables with the results stored in memory. Thus a table look up can replace the reevaluation of the function. Those skilled in the art will recognize that these two methods are functionally equivalent, and will be able to make the time versus space tradeoff when employing an embodiment of the present invention. An equivalent alternative is to start with tables initialized with a permutation of the 256 possible inputs, and perform a key-dependent shuffling of those tables when the key is initialized. Then, during subsequent encryption, a table index operation would be used instead of the current calls to `tbox()`, with equal effect.

[0036] The tables `t1box` and `t2box` are strict permutations, where no entry in the table is equal to its index. This strictness guarantees that there exists no key which is weaker than any other key by allowing an intermediate value in a `tbox()` computation to remain unchanged. The fact that the tables are permutations is important, as described previously in reference to function `tbox()`. If the tables were not permutations, then after the table lookup in the `tbox()` function, there would be some values which could not be the result. These impossible values would allow guesses for return values from `tbox()` and parts of the key to be eliminated, reducing the work to guess the 64 bit key significantly. Alternative embodiments could employ tables which are not permutations, but the encryption would be sub-optimal.

[0037] Any form of cryptanalysis of CMEA must begin by deriving values of the `tbox()` function. A complete analysis, where all outputs for the 256 possible inputs are known, allows CMEA to be applied even without knowing the initial key. However, recovery of the key is possible knowing as few as four distinct values of the function. Thus BEVL places emphasis on disguising the outputs from `tbox()` with other outputs, particularly the value of `tbox(0)`. A number of alternatives are envisioned to accomplish this disguise. The preferred embodiment uses a second different table, `t2box`, and an added pair of passes each which are performed in opposite directions. Any of these three modifications, or sub-combinations thereof, would address the problem to some extent. However, the combination of all three provides the most security.

[0038] In the preferred embodiment, the forward and backward passes use different tables, `t1box` and `t2box`, in conjunction with the `tbox()` function. This is done so that cryptanalysis would require discovery of two complementary sets of function values, rather than just one set. Since the passes tend to disguise each other, two tables provide the best security. Alternative embodiments are envisioned which employ only a single table. While these methods are still secure, they are less secure than those where two tables are employed.

[0039] Begin pass 1 by proceeding from block 99 to block 102, where variable `v` and buffer index `i` are initialized to zero. Then, in block 104, each character `buf[i]` is modified by adding to itself the result of function call `tbox(t1box, v ≈ i)`. The variable `v` is subsequently updated by XORing itself with the new value of `buf[i]`. The buffer index `i` is then incremented. In block 106, if `i < n`, the pass is not complete and flow returns to block 104. When all characters have been modified according to block 104, `i` will equal `n` and pass 1 will be complete. Note that the characters were modified beginning with `buf[0]` working towards the end, `buf[n-1]`.

[0040] Begin pass 2 by proceeding from block 106 to block 202, where variable `v` is initialized to the value `n` and buffer index `i` is initialized to the value `n - 1`. Then, in block 204, each character `buf[i]` is modified by adding to itself the result of function call `tbox(t2box, v ≈ i)`. The variable `v` is subsequently updated by XORing itself with the new value of `buf[i]`. The buffer index `i` is then decremented. In block 206, if `i ≥ 0`, the pass is not complete and the flow returns to block 204. When all characters have been modified according to block 204, `i` will equal `-1` and pass 2 will be complete. Note that,

unlike pass 1, the characters were modified beginning with `buf[n-1]` working towards the beginning, `buf[0]`, and the table `t2box 14` was used instead of table `t1box 12`.

[0041] Pass 3 begins in block 302. Buffer index `i` is initialized to zero. Variable `v` is not used in this pass. Then, in block 304, each character `buf[i]` is modified by XORing with itself the result of function call `tbox(t1box, i+1)`. The buffer index `i` is then incremented. In block 306, if $i < n$, the pass is not complete and the flow returns to block 304. When all characters have been modified according to block 304, `i` will equal `n` and pass 3 will be complete. Note that, like in pass 1, the characters were modified beginning with `buf[0]` working towards the end, `buf[n-1]`, and table `t1box 12` was used. As stated before, however, the direction of pass 3 is not important, since the identical result is achieved with either direction.

[0042] In pass 3, a different output from `tbox()` is combined with each `buf[]` entry. Because the outputs from `tbox()` form a permutation, at most only one such value can possibly be zero. Whether or not there will be a zero depends on the key. In BEVL, the change in the buffer is key-dependent and very difficult to predict. On average, the chance that one of the values will be zero is $n/256$, where `n` is the length of the buffer. Any self-inverting key-dependent or data-dependent change which guarantees that the values in the buffer will be altered is sufficient to ensure encryption. This is an important improvement for BEVL, since, in CMEA, values which remain unchanged lead to cases where the algorithm fails to encrypt at all.

[0043] Begin pass 4 by proceeding from block 306 to block 402, where variable `v` is initialized to `n` and buffer index `i` is initialized to the value `n - 1`. Then, in block 404, a temporary variable `t` is assigned the value returned by the function call `tbox(t2box, v ≈ i)`. The variable `v` is subsequently updated by XORing itself with the current value of `buf[i]`. Each character `buf[i]` is then modified by subtracting from itself the value of temporary variable `t`. The buffer index `i` is then decremented. In block 406, if $i \geq 0$, the pass is not complete and the flow returns to block 404. When all characters have been modified according to block 404, `i` will equal `-1` and pass 4 will be complete. Note that, like in pass 2, the characters were modified beginning with `buf[n-1]` working towards the beginning, `buf[0]`, and table `t2box 14` was used.

[0044] Begin pass 5 by proceeding from block 406 to block 502, where variable `v` and buffer index `i` are initialized to the value zero. Then, in block 504, a temporary variable

t is assigned the value returned by the function call `tbox(t1box, v ≈ i)`. The variable `v` is subsequently updated by XORing itself with the current value of `buf[i]`. Each character `buf[i]` is then modified by subtracting from itself the value of temporary variable `t`. The buffer index `i` is then incremented. In block **506**, if `i < n`, the pass is not complete and the flow returns to block **504**. When all characters have been modified according to block **504**, `i` will equal `n` and pass **5** will be complete. Note that, like in passes **1** and **3**, the characters were modified beginning with `buf[n-1]` working towards the beginning, `buf[0]`, and table `t1box 12` was used.

[0045] Proceed now to block **600**. Encryption is now complete. `Buf[]` now contains the encrypted characters for secure transmission.

[0046] A “C” program implementing the operation described above is provided in FIG. 3. Table `t1box 12` is provided in “C” in FIG. 4. Table `t2box 14` is provided in “C” in FIG. 5.

[0047] The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

WE CLAIM: